M&G No. 60001.0298US01/305246.01

REPRESENTING STYLE INFORMATION IN A MARKUP LANGUAGE DOCUMENT

Related Applications

This patent application is a continuation-in-part application under 35 United States Code § 120 of United States Patent Application No. 10/187,060 filed on June 28, 2002, which is incorporated herein by reference. An exemplary schema in accordance with the present invention is disclosed beginning on page 11 in an application entitled "Mixed Content Flexibility," Serial No. _____, Docket No. 60001.0275US01, filed December 2, 2003, which is hereby incorporated by reference in its entirety.

Background of the Invention

Markup Languages have attained wide popularity in recent years. One type of markup language, Extensible Markup Language (XML), is a universal language that provides a way to identify, exchange, and process various kinds of data. For example, XML is used to create documents that can be utilized by a variety of application programs. Elements of an XML file have an associated namespace and schema.

In XML, a namespace is a unique identifier for a collection of names that are used in XML documents as element types and attribute names. The name of a namespace is commonly used to uniquely identify each class of XML document. The unique namespaces differentiate markup elements that come from different sources and happen to have the same name.

XML Schemata provide a way to describe and validate data in an XML environment. A schema states what elements and attributes are used to describe content in an XML document, where each element is allowed, what types of text contents are allowed within it and which elements can appear within which other elements. The use of schemata ensures that the document is structured in a consistent manner. Schemata

may be created by a user and generally supported by an associated markup language, such as XML. By using an XML editor, the user can manipulate the XML file and generate XML documents that adhere to the schema the user has created. XML documents may be created to adhere to one or more schemata.

The XML standard is considered by many as the ASCII format of the future, due to its expected pervasiveness throughout the hi-tech industry in the coming years. Recently, some word-processors have begun producing documents that are somewhat XML compatible. For example, some documents may be parsed using an application that understands XML. However, much of the functionality available in word processor documents is not currently available for XML documents.

Summary of the Invention

The present invention is generally directed towards a method for representing style information in a markup language (ML) document such as an XML document. Styles provide a method to package a set of formatting (e.g., character formatting, paragraph formatting, table formatting, etc.) under a single user selection that may be used redefined throughout a document.

More particularly, the present invention relates to representing style information in ML so that applications capable of reading a given ML file format, but running in environments where the style information has not been installed, are able to still render the styles. The ML document may be manipulated on a server or anywhere even when the application creating the ML document is not present. Style information (i.e., properties) is saved in a markup language (ML) document without data loss, while allowing the style information to be parsed by ML-aware applications and to be read by ML programmers.

Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention;

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention;

FIGURE 3 illustrates an exemplary portion of an ML file that provides representation of a built-in style applied to a body of a document;

FIGURE 4 illustrates an exemplary portion of an ML file that provides representation of a custom style applied to a body of a document; and

FIGURE 5 shows an exemplary flow diagram for representing style information in a ML document, in accordance with aspects of the invention.

Detailed Description of the Preferred Embodiment

Throughout the specification and claims, the following terms take the meanings explicitly associated herein, unless the context clearly dictates otherwise.

The terms "markup language" or "ML" refer to a language for special codes within a document that specify how parts of the document are to be interpreted by an application. In a word-processor file, the markup language specifies how the text is to be formatted or laid out, whereas in a particular customer schema, the ML tends to specify the text's meaning according to that customer's wishes (e.g., customerName, address, etc). The ML is typically supported by a word-processor and may adhere to the rules of other markup languages, such as XML, while creating further rules of its own.

The term "element" refers to the basic unit of an ML document. The element may contain attributes, other elements, text, and other building blocks for an ML document.

The term "tag" refers to a command inserted in a document that delineates elements within an ML document. Each element can have no more than two tags: the start tag and the end tag. It is possible to have an empty element (with no content) in which case one tag is allowed.

The content between the tags is considered the element's "children" (or descendants). Hence, other elements embedded in the element's content are called "child elements" or "child nodes" or the element. Text embedded directly in the

content of the element is considered the element's "child text nodes". Together, the child elements and the text within an element constitute that element's "content".

The term "attribute" refers to an additional property set to a particular value and associated with the element. Elements may have an arbitrary number of attribute settings associated with them, including none. Attributes are used to associate additional information with an element that will not contain additional elements, or be treated as a text node.

Illustrative Operating Environment

With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device **100**. In a very basic configuration, computing device **100** typically includes at least one processing unit **102** and system memory **104**. Depending on the exact configuration and type of computing device, system memory **104** may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory **104** typically includes an operating system **105**, one or more applications **106**, and may include program data **107**. In one embodiment, application **106** may include a word-processor application **120** that further includes style information **122**. This basic configuration is illustrated in FIGURE 1 by those components within dashed line **108**.

Computing device **100** may have additional features or functionality. For example, computing device **100** may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage **109** and non-removable storage **110**. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory **104**, removable storage **109** and non-removable storage **110** are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other

magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device **100**. Any such computer storage media may be part of device **100**. Computing device **100** may also have input device(s) **112** such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) **114** such as a display, speakers, printer, etc. may also be included. These devices are well know in the art and need not be discussed at length here.

Computing device **100** may also contain communication connections **116** that allow the device to communicate with other computing devices **118**, such as over a network. Communication connection **116** is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

Generally, the present invention is directed at representing style information in an ML document. The ML document may be read by applications that do not share the same schema that created the document.

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention. The exemplary environment shown in FIGURE 2 is a word-processor environment **200** that includes word-processor **120**, ML file **210**, ML Schema **215**, and ML validation engine **225**.

In one embodiment, word-processor **120** has its own namespace or namespaces and a schema, or a set of schemas, that is defined for use with documents associated with word-processor **120**. The set of tags and attributes defined by the schema for word-processor **120** define the format of a document to such an extent that it is referred to as its own native ML. Word-processor **120** internally validates ML file

210. When validated, the ML elements are examined as to whether they conform to the ML schema **215**. A schema states what tags and attributes are used to describe content in an ML document, where each tag is allowed, and which tags can appear within other tags, ensuring that the documentation is structured the same way. Accordingly, ML **210** is valid when structured as set forth in arbitrary ML schema **215**.

ML validation engine **225** operates similarly to other available validation engines for ML documents. ML validation engine **225** evaluates ML that is in the format of the ML validation engine **225**. For example, XML elements are forwarded to an XML validation engine. In one embodiment, a greater number of validation engines may be associated with word-processor **120** for validating a greater number of ML formats.

Representing Style Information in a Markup Language Document

Styles may be applied to paragraphs, characters, tables, lists, and other structures within a document. An individual style may include a specified font size, bolding, orientation, indentation, color, and other formatting. In an ML file, the styles declarations are listed at the top, prior to the body of the document. The declarations set up the styles so that each style may be referenced by the different objects in the body of the document. In one embodiment, a root element "w:wordDocument" allows a child element "w:styles" which stores the style definitions. The children of the styles element may include a version of a built in style, a latent style, and a custom style (e.g., a user generated style).

There may be a number of styles built in directly into a word processor application. These built-in styles are used by many of application's internal features. Since the styles are built in, the application may not be required to view the definition of the styles when the application opens a document, but it usually will. It is still possible for a user to change the appearance of a built in style, so it is usually necessary to look a the properties of the build in style when opening the file. If there is no definition though, the application knows how the built-in style should look.

However, future versions of the built-in style may change the name with which the style is referred. With the version number for built-in style names, the application knows whether a style is a built in style using an older version name, or if it is a custom style. If the style is one of the built in styles, but it's just using an old name, than the current application can change the name of the style and update all objects that are referencing that old name. An example of a use of this would be if in a new version there were an improvement made to Heading styles. With the improvement the name of Heading styles were changed. In order for old documents to still have their Headings treated as headings and not just custom styles, the version of built in style names is referenced. This is also useful in international scenarios where a built in style may be called something else in a different language (although this is not usually the case with built in styles).

The "style" element includes the definition of a particular style. Each style definition is included in it's own "style" element which is a child of the "styles" element. The style element includes a number of elements and attributes, of which some examples are described in the following table:

name	*Primary name of style; built-in stylename are converted to a language-independent form*
aliases	*Secondary names of style, separated by commas*
ref = "wx:uiName"	*This is just a hint. It shows the name of the style as it appears in the UI*
sti	*Built-in style unique numerical identifier*
basedOn	*styleId name of style this style is based on*
next	*styleId name of the next-paragraph-style, only for paragraph styles*
link	*styleId of the linked style, only for linked paragraph and character*

	styles
rsid	*Revision Save Id for this style, a unique identifier to track when the style was last changed*
@styleId	*Name used to refer to this style within ML. Unique within the file. Otherwise unused.*
pPr	*Paragraph properties for the style, if any*

Table 1

The elements associated with the "style" element each have a specific function associated with the "style" element. For example, the <name> element is used to show what a particular style is called when the style is shown in a word-processing application. The <name> element is different from the styleID (i.e., *@styleId*). The style ID is what the different objects in the document that are part of that style use to refer to the style. Typically, the styleID element is syntactically formed in accordance with CSS semantics (used in HTML documents), which is usually more restrictive than the rules of the application (e.g., a word processor or a spreadsheet). Using styleIDs with CSS-compatible semantics results in reduced parsing for the application when the application transforms the document into ML code.

Another example element is pPr element, or the paragraph element. The pPr element includes information on the paragraphs properties for the given style. The paragraph properties may include a variety of properties, of which some examples are described in the following table:

listPr	**[listPrElt]**	*List properties*
supressLineNumbers	**[onOffProperty]**	*Prevents line numbers from appearing next to paragraph. This setting has no effect in documents or sections with no line numbers.*
pBdr	**[pBdrElt]**	*Borders for the paragraph*

shd	**[shdProperty]**	*Paragraph Shading*
tabs	**[tabsElt]**	*Tab Stop List*
suppressAutoHyphens	**[onOffProperty]**	*Prevents automatic hyphenation*
kinsoku	**[onOffProperty]**	*Asian Typography: Use East Asian typography and line-breaking rules to determine which characters begin and end a line on a page.*
wordWrap	**[onOffProperty]**	*Asian Typography: Allows a line to break in the middle of a Latin word.*
overflowPunct	**[onOffProperty]**	*Asian Typography: Allows punctuation to continue one character beyond the alignment of other lines in the paragraph. If you do not use this option, all lines and punctuation must be perfectly aligned.*
topLinePunct	**[onOffProperty]**	*Asian Typography: Allows punctuation to compress at the start of a line, which lets subsequent characters move in closer.*

Table 2

These properties add to the functionality provided for styles in ML, allowing the styles to be mapped to an ML file and substantially reproduced when parsed by an application.

As may be seen in the examples in FIGURES 3 and 4 below, unless a style is specified, a default style (e.g., normal) is applied. Usually, a style is based on a previous style included in the style definitions for the ML document. Most often a new style created is based on the normal style, meaning that new style inherits all of the properties from the Normal style, and then builds on those. It is also possible though to base a style on another style that already has the desired base appearances. This allows for an easy changing of the "theme" of the entire document, since all the styles can be based on one core style that can then be modified to affect the entire document's look.

FIGURE 3 illustrates an exemplary portion of an ML file that provides representation of a built-in style applied to a body of a document, in accordance with aspects of the present invention.

The example shown, illustrates a styles definition section **310** and a body **330** of the ML file **300**. Styles definition section **310** includes a default built-in style "Normal" **320**. The style is of type "paragraph" but may also be another type of style, such as table or list. Normal style **320** shown has associated elements such the rPr element that includes the character properties associated with the paragraph style. The result of the ML document is the typed word "Hello" **340**, as shown the body **320** of the ML file **300**. Since normal style **320** is the default style defined for the document, a reference to normal style **320** is not required in the body **320** of the ML file **300**. Instead, the default style is automatically applied to text of the body **320**. It is appreciated that a variety of styles may be associated with the ML file **300** along with a default style for each style type (e.g., a default table style).

FIGURE 4 illustrates an exemplary portion of an ML file that provides representation of a custom style applied to a body of a document, in accordance with aspects of the present invention.

The example of FIGURE 4 is similar to the example of FIGURE 3 illustrating a styles definition section **410** and a body **440** of the ML file **400**. In addition, the example of FIGURE 4 includes custom style "MyStyle" **430** along with default built-in style "Normal" **420** defined within styles definition section **410**, and additional result "My special sentence" **470** along with result "Plain" **450** in the body **440**. A reference the body **460** to Mystyle **430** applies the custom style to the result that follows the reference. Other custom styles may be defined and applied to results within ML document **400**. A variety of styles may be defined using the custom styles functionality of the present invention and applied to multiple results throughout a document.

The following is an exemplary portion of schema for generating the styles in the ML document, in accordance with aspects of the present invention:

```
<xsd:element name="styles" type="stylesElt" minOccurs="0">
    <xsd:annotation>
```

<xsd:documentation>Represents the style definitions.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:complexType name="stylesElt">
<xsd:annotation>
<xsd:documentation>Defines a collection of styles.</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="versionOfBuiltInStylenames" type="decimalNumberProperty" minOccurs="0">
<xsd:annotation>
<xsd:documentation>Represents the version of the built-in style names. If the name of a built-in style changes, this version number is incremented.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="latentStyles" type="latentStylesElt" minOccurs="0" maxOccurs="1">
<xsd:annotation>
<xsd:documentation>Represents information about latent (not-yet-instantiated built-in) styles.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="style" type="styleElt" minOccurs="0" maxOccurs="unbounded">
<xsd:annotation>
<xsd:documentation>Represents the style definition.</xsd:documentation>

```
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="latentStylesElt">
            <xsd:annotation>
                <xsd:documentation>Defines a specific instance of a latent style (the attributes give information about the locked state).</xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="lsdException" type="lsdExceptionElt" minOccurs="0" maxOccurs="unbounded">
                    <xsd:annotation>
                        <xsd:documentation>Represents a specific instance of a latent style (the attributes give information about the locked state).</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="defLockedState" type="onOffType">
                <xsd:annotation>
                    <xsd:documentation>Gets or sets defines the default locked state.</xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="latentStyleCount" type="decimalNumberType">
                <xsd:annotation>
                    <xsd:documentation>If not zero, represents the number of latent styles that should be initialized to the given defaults.</xsd:documentation>
```

```
            </xsd:annotation>
        </xsd:attribute>
    </xsd:complexType>

    <xsd:complexType name="styleElt">
        <xsd:annotation>
            <xsd:documentation>Defines a document style.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="name" type="stringProperty" minOccurs="0"
maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation>Represents the primary name of
style. Built-in style names are converted to a language-independent
form.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="aliases" type="stringProperty" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>Represents the secondary names of
a style, separated by commas</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element ref="wx:uiName" minOccurs="0" maxOccurs="1">
                <xsd:annotation>
                    <xsd:documentation>A hint as to the name for this style
as it appears to the user in Word.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
```

```
                <xsd:element name="sti" type="decimalNumberProperty"
minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Represents the built-in style's
unique numerical identifier.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="basedOn" type="stringProperty" minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Represents the styleId (name of
style) this style is based on.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="next" type="stringProperty" minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Represents the styleId name of the
next-paragraph-style; used only for paragraph styles.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="link" type="stringProperty" minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Represents the styleId of the linked
style; used only for linked paragraph and character styles.</xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="autoRedefine" type="onOffProperty"
minOccurs="0">
                    <xsd:annotation>
                        <xsd:documentation>Specifies whether this style can be
automatically redefined when appropriate.</xsd:documentation>
```

```
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="hidden" type="onOffProperty" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>Specifies whether to show this style to the user.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="semiHidden" type="onOffProperty" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>Specifies not to show this style to users unless they request to see it.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="locked" type="onOffProperty" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>Restricts this style from use by the end user.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="personal" type="onOffProperty" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation>Specifies whether this is another user's HTML threading personal style.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="personalCompose" type="onOffProperty" minOccurs="0">
                <xsd:annotation>
```

<xsd:documentation>Specifies whether this is another user's HTML threading compose style.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="personalReply" type="onOffProperty" minOccurs="0">
<xsd:annotation>
<xsd:documentation>Specifies whether this is another user's HTML threading reply style.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="rsid" type="longHexNumberProperty" minOccurs="0">
<xsd:annotation>
<xsd:documentation>Represents the Revision Save ID for this style, which is a unique identifier used to track when the style was last changed.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="pPr" type="pPrElt" minOccurs="0" maxOccurs="1">
<xsd:annotation>
<xsd:documentation>Represents the paragraph properties for the style, if any.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="rPr" type="rPrElt" minOccurs="0" maxOccurs="1">
<xsd:annotation>

<xsd:documentation>Represents the character properties for the style, if any.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="tblPr" type="tblPrElt" minOccurs="0" maxOccurs="1">
<xsd:annotation>
<xsd:documentation>Represents the table properties for the style.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="trPr" type="trPrElt" minOccurs="0" maxOccurs="1">
<xsd:annotation>
<xsd:documentation>Represents the row properties for the style.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="tcPr" type="tcPrElt" minOccurs="0" maxOccurs="1">
<xsd:annotation>
<xsd:documentation>Represents the cell properties for the style.</xsd:documentation>
</xsd:annotation>
</xsd:element>
<xsd:element name="tblStylePr" type="tblStylePrElt" minOccurs="0" maxOccurs="unbounded">
<xsd:annotation>
<xsd:documentation>Represents the table-style conditional-override properties.</xsd:documentation>

```
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="type" type="styleKindValue" use="optional">
                <xsd:annotation>
                    <xsd:documentation>Gets or sets the type of
style.</xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="styleId" type="stringType" use="optional">
                <xsd:annotation>
                    <xsd:documentation>Gets or sets the name used to refer to this
style within XML. This name is unique within the file. This attribute is otherwise
unused.</xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="default" type="onOffType" use="optional">
                <xsd:annotation>
                    <xsd:documentation>Specifies whether this style is the default
for this type of style.</xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
</xsd:complexType>
```

FIGURE 5 shows an exemplary flow diagram for representing style information in a ML document, in accordance with aspects of the invention. After start block **510**, the process flows to block **520** where the style information within a document such as a word-processor document, is determined. The style information used within a document may include many different types of styles such as paragraph styles, table styles, character styles, and list styles, including those that are not natively

supported by later applications parsing the document. Once the style information is determined, processing proceeds to decision block **530**.

At block **530**, the properties of the styles within the document are mapped into elements, attributes, and values of the ML file. The styles and the properties associated with the styles may change from page to page, section to section, chapter to chapter and the like. There may be more than one mapping, therefore, per document. Once the style information properties are mapped, or written to the ML file, processing moves to block **540**.

At block **540**, the properties of the styles are stored in a ML document that may be read by applications that understand the ML. Once the properties are stored, processing moves to end block **550** and returns to processing other actions.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.